BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

PATENT ABSTRACTS OF JAPAN

(11)Publication number : **06-250847**
(43)Date of publication of application : **09.09.1994**

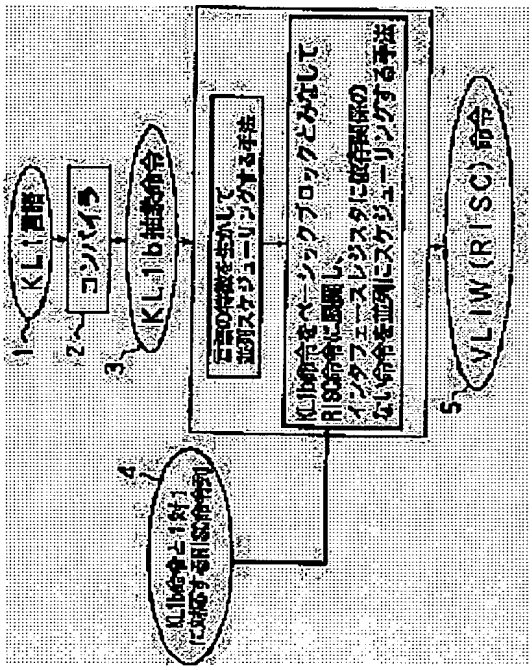
(51)Int.Cl. **G06F 9/45**
G06F 9/44
G06F 15/16

(21)Application number : **05-036601**
(22)Date of filing : **25.02.1993**

(71)Applicant : **FUJITSU LTD**
(72)Inventor : **SAKAMOTO MARIKO**

(54) CONVERSION SYSTEM FOR CONVERTING INSTRUCTION OF PARALLEL LOGIC TYPE LANGUAGE INTO VLIW TYPE INSTRUCTION

(57)Abstract:
PURPOSE: o provide a conversion system for converting an instruction of parallel logic type language such as GHC and KL1 into an VLIW instruction to be executed by a VLIW type parallel computer.
CONSTITUTION: A parallel logic type language 1 such as a KL1 language is compiled by a compiler 2 to generate a KL1b abstract instruction 3. Then the instruction 3 is scheduled by utilizing the feature of the parallel logic type language. On the other hand, the parallel pattern 4 of an RISC instruction string corresponding to the instruction 3 at the rate of 1 to 1 is previously set up and the instruction 3 is developed to an RISC instruction string. In this case, the depending relation of an interface register is checked by regarding the instruction 3 as a basic block and the instruction 3 having no depending relation is scheduled in parallel.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号

特開平6－250847

(43)公開日　平成６年(1994)９月９日

(51)Int.Cl.[5]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	9/45					
	9/44	３３０　Ｂ	9193－5B			
	15/16	４３０	9190－5L			
			9292－5B	Ｇ０６Ｆ　9/44	３２２　Ｆ	

審査請求　未請求　請求項の数４　ＯＬ　（全 17 頁）

(21)出願番号　特願平5－36601

(22)出願日　平成５年(1993)２月25日

(71)出願人　000005223
富士通株式会社
神奈川県川崎市中原区上小田中1015番地

(72)発明者　坂本　真理子
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内

(74)代理人　弁理士　京谷　四郎

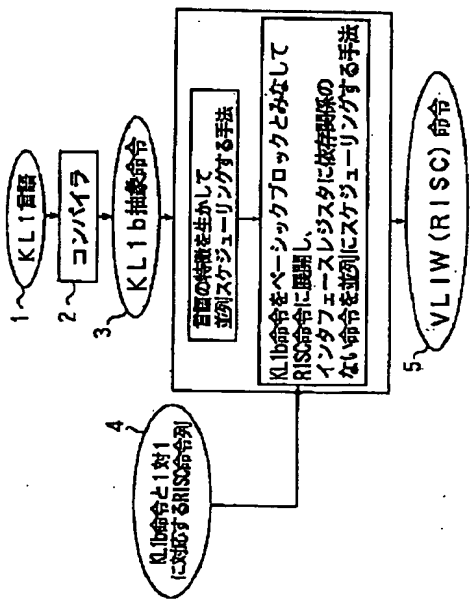
(54)【発明の名称】　並列論理型言語の命令をＶＬＩＷ方式の命令に変換する変換方式

(57)【要約】

【目的】　ＧＨＣ，ＫＬ１などの並列論理型言語の命令を、ＶＬＩＷ方式の並列計算機で実行できるようなＶＬＩＷ命令に変換する変換方式を提供すること。

【構成】　ＫＬ１言語などの並列論理型言語１をコンパイラ２によりコンパイルしてＫＬ１ｂ抽象命令３を生成する。ついで、並列論理型言語の特徴を生かしてＫＬ１ｂ抽象命令３を並列にスケジューリングする。一方、ＫＬ１ｂ抽象命令３に一対一に対応するＲＩＳＣ命令列の並列パターン４を設定しておき、ＫＬ１ｂ抽象命令をＲＩＳＣ命令列に展開する。その際、ＫＬ１ｂ抽象命令３をベーシック・ブロックとみなして、インタフェース・レジスタの依存関係を調べ、依存関係のないＫＬ１ｂ抽象命令３を並列にスケジューリングする。

本発明の原理説明図



1

【特許請求の範囲】

【請求項１】　抽象度の高い並列論理型言語の命令を、命令レベルで並列実行を行うＶＬＩＷ方式の並列計算機で実行できる機械語レベルに近いＲＩＳＣ命令列に変換する変換方式において、

並列論理型言語をコンパイルして、並列論理型言語の命令を生成し、

生成された並列論理型言語の命令を並列論理型言語の特徴を生かして並列にスケジューリングし、

スケジューリングされた並列論理型言語の命令を、並列論理型言語の命令に一対一に対応したＲＩＳＣ命令列に展開することにより、並列論理型言語をＶＬＩＷ方式の命令に変換することを特徴とする並列論理型言語の命令をＶＬＩＷ方式の命令に変換する変換方式。

【請求項２】　並列論理型言語の命令に一対一に対応させて、ＲＩＳＣ命令を展開し、展開されたＲＩＳＣ命令列の並列性を解析して、並列に実行できる命令を横に並べて記述することにより、あらかじめ、並列論理型言語の命令に一対一に対応した、ＲＩＳＣ命令列の並列パターンを設定しておくことを特徴とする請求項１の並列論理型言語の命令をＶＬＩＷ方式の命令に変換する変換方式。

【請求項３】　並列論理型言語の命令をベーシック・ブロックと見なし、ベーシック・ブロックの融合の可否を並列論理型言語の仕様にある並列性とインタフェース・レジスタの依存関係に基づき判定することにより、並列論理型言語の命令を並列に記述することを特徴とする請求項１の並列論理型言語の命令をＶＬＩＷ方式の命令に変換する変換方式。

【請求項４】　ベーシック・ブロック融合する際に、ベーシック・ブロックの中の、並列に実行される可能性のある並列論理型言語の命令を実行するＲＩＳＣ命令の依存関係を調査することなく、並列なＲＩＳＣ命令列を生成することを特徴とする請求項３の並列論理型言語の命令をＶＬＩＷ方式の命令に変換する変換方式。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、抽象度の高い並列論理型言語の命令をＶＬＩＷ方式の計算機で高速に実行するため、並列論理型言語の命令をＶＬＩＷ命令に変換する変換方式に関する。並列論理型言語での並列処理は、後述するように複数のゴールを同時に実行する方式で実現されている。それに対して、ＶＬＩＷ方式の並列処理は、ゴールより機械命令に近いレベルで命令を同時に実行して実現される。

【０００２】本発明は抽象度の高い並列論理型言語の命令を、より機械語に近いレベルであるＲＩＳＣ命令に展開し、また、その展開の過程で並列化のスケジューリングを行う、並列論理型言語の命令をＶＬＩＷ命令に変換する変換方式に関するものである。

2

【０００３】

【従来の技術】従来、ＧＨＣ、ＫＬ１などの並列論理型言語は単一の逐次実行プロセッサ、あるいは、逐次実行のプロセッサを複数接続したシステムで実行されていた。プロセッサを複数接続したシステムでは、いわゆる並列実行が実現されていたが、この場合の並列実行はゴール・レベルでの並列実行であり、命令レベルの並列実行は実現されていなかった。

【０００４】

【発明が解決しようとする課題】本発明は、上記した従来技術を考慮して発明されたものであって、ＧＨＣ，ＫＬ１などの並列論理型言語の命令を、ＶＬＩＷ方式の並列計算機で実行できるようなＶＬＩＷ命令に変換する変換方式を提供することを目的とする。

【０００５】

【課題を解決するための手段】図１は本発明の原理説明図である。上記課題を解決するため、本発明の請求項１の発明は、図１に示すように、抽象度の高い、例えば、ＫＬ１言語のＫＬ１ｂ抽象命令などの並列論理型言語の命令を、命令レベルで並列実行を行うＶＬＩＷ方式の並列計算機で実行できる機械語レベルに近いＲＩＳＣ命令列に変換する変換方式において、並列論理型言語ＫＬ１をコンパイルして、並列論理型言語の命令ＫＬ１ｂを生成し、生成された並列論理型言語の命令ＫＬ１ｂを並列論理型言語の特徴を生かして並列にスケジューリングし、スケジューリングされた並列論理型言語の命令ＫＬ１ｂを、並列論理型言語の命令ＫＬ１ｂに一対一に対応したＲＩＳＣ命令列に展開することにより、並列論理型言語をＶＬＩＷ方式の命令に変換するようにしたものである。

【０００６】本発明の請求項２の発明は、請求項１の発明において、並列論理型言語の命令ＫＬ１ｂに一対一に対応させてＲＩＳＣ命令を展開し、展開されたＲＩＳＣ命令列の並列性を解析して、並列に実行できる命令を横に並べて記述することにより、あらかじめ、並列論理型言語の命令ＫＬ１ｂに一対一に対応した、ＲＩＳＣ命令列の並列パターンを設定しておくようにしたものである。

【０００７】本発明の請求項３の発明は、請求項１の発明において、並列論理型言語の命令ＫＬ１ｂをベーシック・ブロックと見なし、ベーシック・ブロックの融合の可否を論理型言語の仕様にある並列性とインタフェース・レジスタの依存関係に基づき判定することにより、論理型言語の命令ＫＬ１ｂを並列に記述するようにしたものである。

【０００８】本発明の請求項４の発明は、請求項３の発明において、ベーシック・ブロックを融合する際に、ベーシック・ブロックの中の、並列に実行される可能性のある並列論理型言語の命令を実行するＲＩＳＣ命令の依存関係を調査することなく、並列なＲＩＳＣ命令列を作

成するようにしたものである。

【0009】

【作用】図1において、例えば、ＫＬ１言語などの並列論理型言語１をコンパイラ２によりコンパイルして並列論理型言語のＫＬ１ｂ抽象命令３を生成する。ついで、並列論理型言語の特徴を生かしてＫＬ１ｂ抽象命令３を並列にスケジューリングする。

【0010】一方、ＫＬ１ｂ抽象命令３に一対一に対応するＲＩＳＣ命令列の並列パターン４を設定しておき、ＫＬ１ｂ抽象命令３をベーシック・ブロックとみなして、ＫＬ１ｂ抽象命令をＲＩＳＣ命令列に展開する。また、その際、インタフェース・レジスタの依存関係を調べ、依存関係のないＫＬ１ｂ抽象命令３を並列にスケジューリングする。

【0011】本発明においては、上記のようにして、抽象度の高いＫＬ１ｂ抽象命令などの並列論理型言語の命令を、ＶＬＩＷ方式の並列計算機で実行できるＲＩＳＣ命令列に変換しているので、複雑な最適化処理を行うことなく、並列論理型言語をＲＩＳＣ命令レベルで並列化することができ、容易にＶＬＩＷ方式の命令を作成することができる。

【0012】

【実施例】まず、本発明の前提となるＶＬＩＷ方式および並列論理型言語の特徴について説明する。

１．１　ＶＬＩＷ方式について

ＶＬＩＷ方式の命令とは、コンパイラが出力するコード形式の一つであり、ＶＬＩＷコードを生成するコンパイラが出力する命令がＶＬＩＷ命令と呼ばれる。

【0013】図２はＶＬＩＷ命令の形を示す図であり、同図に示すように、ＶＬＩＷ命令の内部には、複数個の逐次実行命令が並列に記されている。そして、ＶＬＩＷ方式の並列計算機はＶＬＩＷ命令を実行の単位として、同図の１，２，…，ｉ，…，ｍの順序で命令を実行する。これにより、各ＶＬＩＷ命令の内部に並列に記されている複数個の逐次命令１ないしｎが同時に実行される。

１．２　並列論理型命令の特徴

ａ）並列論理型言語の命令は抽象レベルの高い命令である。

```
goukei(A,B,C,D,Z):-true    | add(A,B,X),add(C,D,Y),add(X,Y,Z).
add(X,Y,Z):-true           | Z=X+Y.
```

プログラムはゴールが与えられて起動される。上記例において、プログラムの起動をgoukei(1,2,3,4,Z) というゴールで行う。まず、与えられたゴール名と等しいクローズヘッドを持ったクローズが選択され、引数の個数と型があっているかチェックされる。goukeiの場合、与えられたゴールの引数が５個で４つの引数が整数であるので、クローズが選択されるためには、ヘッド部のゴールの引数が与えられた値と等しい数値か、もしくは、構造を持たない変数でなければならない。

【0019】並列論理型言語の言語仕様にある並列性の特徴の一つはここにあり、①クローズを選択する時に行われる引数の判定は、複数の引数に対して同時に実行できる。この並列性を、以降、ヘッド部のユニフィケーション（ヘッド部における型の判定）と言う。上記例では、プログラムのgoukeiの引数は全て構造を持たない変数であり、与えられたゴールと引数と個数があっているので、goukeiのクローズが選択される。　すなわち、上記プログラムにおいて、goukei(A,B,C,D,Z) の引数の型はgoukei(1,2,3,4,Z) と同様、ＬＩＳＴ形式などの構造を持つ変数でなく、その個数も５個なので、このクロー

＊る。

【0014】並列論理型言語のコンパイラが出す、ゴール・レベルで並列実行が可能な命令は、抽象度の高い命令である（以降、並列論理型言語のコンパイル結果をＫＬ１ｂ、または、並列論理型言語の命令という）。この命令をＶＬＩＷに列挙できる機械語命令に近いレベルの命令（以下、このような命令をＲＩＳＣ命令という）に展開することができる。

【0015】そして、並列論理型言語の個別の命令を、ＲＩＳＣ命令列で置き換えるため、一般には、このＲＩＳＣ命令列は入口／出口がそれぞれ一つずつの命令列であるベーシック・ブロックとみなすことが可能であり、このブロック内で並列化スケジューリングを行うことができ、このブロック内のスケジューリングは、静的な情報（実行結果を含まない情報）のみで命令を並列化できる。したがって、並列論理型言語の命令は１対１で並列化スケジューリング済みのＲＩＳＣ命令列と対になる。

ｂ）並列論理型言語の仕様に並列性が存在する。

【0016】並列論理型言語には、その名前が示すように、言語の仕様として並列性が存在する。本実施例においては、一例として、並列論理型言語ＫＬ１の並列性について説明する。ＫＬ１はフラットＧＨＣ（Flat Guarded Horn Clauses)に基づいて設計された言語であり、以下、言語の仕様にある並列性を、ＫＬ１プログラムの実行の様子と合わせて述べる。

【0017】ＫＬ１プログラムは次のシンタックスを持つクローズの集合として表される。

Ｈ：－Ｇ1,…，Ｇm ｜Ｂ1,…，Ｂn.（ｍ≧０，ｎ≧０）

ここで、Ｈ，Ｇｉ，Ｂｊは各々クローズ・ヘッド、ガード・ゴール、ホディー・ゴールと呼ばれる。オペレータ「｜」はコミットメントバーと呼ばれ、クローズの中でこれに先立つ部分をガード部、これに続く部分をボディ部と呼ぶ。ここで、ガードゴールとしては、組み込み述語しか書けない。また、ボディゴールには組み込み述語とユーザ定義述語の２種類が書ける。

【0018】次に、上記ＫＬ１のプログラムがどのようにして実行されるか、下記のプログラムについて簡単に説明する。

5

ズが選択される。

【００２０】クローズが選択されると、ガード部の述語が実行される。並列論理型言語の言語仕様にある第２の並列性は、②ガード部の複数の述語は同時に実行できるという点にある。しかしながら、上記プログラムにおいては、ガード部は常に成立する述語（true：常に真であるので次を実行）が与えられているので、実処理はなく、すぐにボディー部が実行される。

【００２１】ボディー部には、ユーザ定義述語のａｄｄが３回使われている。並列論理型言語の言語仕様にある第３の並列性は、③ボディー部の述語はすべて同時に実行することが可能であるという点にある。したがって、上記例におけるａｄｄは同時に実行できる（これを子ゴールの生成という）。ただし、３つめのａｄｄは１つめと２つめの結果を使用しているので、実際は１つめと２つめのａｄｄの結果が決まるのを待って実行される。このように、データの依存関係によって、同時に実行できない場合がある。

【００２２】ボディー部の述語ａｄｄは上記goukeiと同じように新しいクローズを起動させるゴールとなって、同じゴール名のクローズの選択に移る。例えば、上記例におけるadd(A,B,C)というゴール名と等しいコール名のクローズ・ヘッドを持ったクローズ（次の行の add(X,Y,Z)）が選択される。ここでも、引数の個数と型があっているかの判定が同時に行われ、ａｄｄのボディー部では、演算の結果をＺに代入する際にユニフィケーション（型の判定）が実行される。

【００２３】ここで、④複数のユニフィケーションが存在する場合には、それらは同時に実行でき、これは並列論理型言語の言語仕様にある第４の並列性である（これを、ボディー部のユニフィケーションという）。次に、並列論理型言語ＫＬ１のコンパイル・コードであるＫＬ１ｂをＶＬＩＷ命令列に変換する手順について説明する。

２．１　ＫＬ１ｂのＲＩＳＣ命令列への展開

ＫＬ１ｂは抽象度の高い命令である。したがって、ＫＬ１ｂ命令をＲＩＳＣ命令で書き直すと、１つのＫＬ１ｂ命令は複数のＲＩＳＣ命令で置き換えられることになる。また、１つ１つのＫＬ１ｂ命令は当然ながら、入口／出口がそれぞれ一つなので、ＦＯＲＴＲＡＮ等でいうベーシック・ブロックと考えることができる。

【００２４】したがって、各々のＫＬ１ｂ命令単位で対応するＲＩＳＣ命令列を静的にスケジュールすることが可能であり、図３に示すように、ＫＬ１ｂ命令とＲＩＳＣ命令列の対を生成することができる。例えば、ＫＬ１ｂ命令の機能をＲＩＳＣ命令を用いてコーディングすることにより、ＫＬ１ｂ命令に対応するＲＩＳＣ命令列を生成することができる。

【００２５】上記のように、ＫＬ１ｂ命令と対応づけ、ＲＩＳＣ命令レベルでスケジューリングを施したＲＩＳＣ命令の組をデータとして用意すると、ＫＬ１ｂ命令を並列ＲＩＳＣ命令列に自動的に変換するシステムを得ることができる。

6

２．２　同時に実行可能なＫＬ１ｂ命令の並列記述

並列論理型言語の特徴から同時に実行可能なＫＬ１ｂ命令を記述する。図４は同時に実行可能なＫＬ１ｂ命令を並列に記述した一例を示す図であり、同図に示すように、並列論理型言語の特徴から同時に実行可能なＫＬ１ｂ命令を並列に記述することができる。なお、同時に実行可能なＫＬ１ｂ命令を並列に記述する際には、データの依存関係をこわさないような注意が必要である。ここで、同時に実行可能だとみなされたＫＬ１ｂ命令は、ＦＯＲＴＲＡＮ等のトレース・スケジューリングの最適化でいう、ベーシック・ブロックを広げてトレースを大きくとった（２以上のベーシック・ブロックをつなげて１つとする）のと同じ状態となる。

【００２６】また、上記スケジューリングを行う際には、インタフェース・レジスタを調査して並列の可能性の可否を判断する。次に示すプログラムは、ＫＬ１ｂ言語で書かれたａｐｐｅｎｄプログラムの命令列の一例を示しており、次のプログラムにより並列性の可否の判断について説明する。

【００２７】

```
label  append/3.
         ...
       put__list 5.                    (0)
       write__car_value 5,4            (1)
       write__cdr_ variable 5,4        (2)
       get__list__ value 5,4
       put__value 4,3
       execute append/3.
```

上記例は、（０）でリスト構造が作られ、（１）と（２）でリストのcar 部とcdr 部にデータが書き込まれる命令列である。

【００２８】上記例の（１）の write__car_value 5,4 はインタフェース・レジスタ５が持つリスト構造のcar部にインタフェース・レジスタ４にあるデータを書き込むことを指示する命令である。また、（２）の write__cdr_ variable 5,4 はインタフェース・レジスタ５が持つリスト構造のcdr 部とインタフェース・レジスタ４の両方から、新しいデータ領域を指すことを指示する命令である。すなわち、インタフェース・レジスタ５のcdr部とインタフェース・レジスタ４に新しい変数を指示するアドレスを格納し、新しいデータ領域を指すことを指示する命令である。

【００２９】上記命令（０），（１），（２）においては、インタフェース・レジスタ５を使用する。これをインタフェース・レジスタ５にぶつかりがあるという。ここで、命令（１）と（２）はレジスタ５を参照するので、インタフェース・レジスタ５にデータを書き込む命

令（０）が終了するのを待って実行されなければならない。また、（１）と（２）の命令でインタフェース・レジスタ４にぶつかりがあるが、命令（１）と命令（２）を並列に並べても、命令の実行に際しては、参照のあとで書き込みが生じ、（２）でインタフェース・レジスタ５に書き込みが生ずるのは（１）の命令でインタフェース・レジスタ４を参照したのちであるので、この二つの命令を同時に実行するのは問題がない。

２．３　並列に記述されたＫＬ１ｂのＲＩＳＣ命令での置き換え。

【００３０】上記２．１で述べたように、各々のＫＬ１ｂ命令はＲＩＳＣ命令列に展開することができ、また、上記２．２で述べたようにＫＬ１ｂ命令はそれらの間に依存関係がなく並列に記述されているから、並列に記述されたＫＬ１ｂ命令をシステムにより自動的にＲＩＳＣ命令列に展開し、ＲＩＳＣ命令列の間の依存関係を調べることなく並列のＲＩＳＣ命令列を生成することができる。

【００３１】図５は並列に実行可能なＫＬ１ｂ命令１、ＫＬ１ｂ命令２、および、ＫＬ１ｂ命令３を、それぞれのＫＬ１ｂ命令に対応したＲＩＳＣ命令列に置き換えた場合を示した図であり、同図に示すように並列に記述されたＲＩＳＣ命令列を得ることができる。

２．４　ＲＩＳＣ命令の整理

図５に示す広げられたベーシック・ブロックは、複数のＲＩＳＣ命令の組が並べて置かれた状態になるので、ブロック中でＶＬＩＷの並列度に近くなるように、ＲＩＳＣ命令列を整理する。

【００３２】すなわち、ＲＩＳＣ命令の内、横に並べかえることができる命令を横に並びかえ、実質的な並列度を高くする。その結果、図６に示すように、並列論理型言語の命令を、ＶＬＩＷ方式の並列計算機で実行できるようなＶＬＩＷ命令に変換することができる。次に、下記に示す、ゴールの引数として与えられた２つのリストをつなげる処理を行うＫＬ１で記述されたａｐｐｅｎｄというプログラムを一例として、具体的な実施例について説明する。

```
append([A | X], Y, Z):-true | Z=[ A | Z 1], ap
end(X, Y, Z 1).
append([], Y, Z):-true | Z=Y.
```

上記プログラムをコンパイルすると、図７に示すコードのＫＬ１ｂ命令列が得られる。

＜実施例＞次に、前記２．１で説明したように、図７に示すＫＬ１ｂ命令を対応するＲＩＳＣ命令列に変換する。なお、図７に示したＫＬ１ｂ命令列に対応したＲＩＳＣ命令を示すと膨大な量になるので、上記ａｐｐｅｎｄというプログラムに現れたＫＬ１ｂ命令に対して、対となるＲＩＳＣ命令を示すと図８ないし図１１に示すようになる（図３に対応）。

【００３３】一方、上記ａｐｐｅｎｄというプログラム

のＫＬ１ｂ命令列には図１２に示すような並列性がみられる（図４に対応）。そこで、図１２に示すＫＬ１ｂ命令をＲＩＳＣ命令で置き換えて、ここでは、ＶＬＩＷの並列度を３として、ＲＩＳＣ命令列を整理すると、図１３ないし図１６に示すＲＩＳＣ命令列を得ることができる。

【００３４】

【発明の効果】以上説明したように、本発明においては、抽象度の高いＫＬ１ｂ抽象命令などの並列論理型言語の命令を、並列論理型言語の特徴を生かして並列にスケジューリングし、スケジューリングされた並列論理型言語の命令を、並列論理型言語の命令に一対一に対応した機械語レベルに近いＲＩＳＣ命令列に展開することにより、ＶＬＩＷ方式の並列計算機で実行できるＲＩＳＣ命令列に変換しているので、複雑な最適化処理を行うことなく、並列論理型言語をＲＩＳＣ命令レベルで並列化することができ、容易にＶＬＩＷ方式の命令を作成することができる。

【図面の簡単な説明】

【図１】本発明の原理説明図である。

【図２】ＶＬＩＷ命令の形を示す図である。

【図３】ＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図である。

【図４】ＫＬ１ｂ命令の逐次実行列を並列に変換した状態を示す図である。

【図５】ＫＬ１ｂ命令をＲＩＳＣ命令列に展開した状態を示す図である。

【図６】広げられたベーシック・ブロックの中のＲＩＳＣ命令を示す図である。

【図７】実施例のコンパイル結果を示す図である。

【図８】実施例のＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図である。

【図９】実施例のＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図（続き）である。

【図１０】実施例のＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図（続き）である。

【図１１】実施例のＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図（続き）である。

【図１２】実施例のＫＬ１ｂ命令を並列に記述した状態を示す図である。

【図１３】実施例をＲＩＳＣ命令列に置き換え整理した状態を示す図である。

【図１４】実施例をＲＩＳＣ命令列に置き換え整理した状態を示す図（続き）である。

【図１５】実施例をＲＩＳＣ命令列に置き換え整理した状態を示す図（続き）である。

【図１６】実施例をＲＩＳＣ命令列に置き換え整理した状態を示す図（続き）である。

【符号の説明】

１　ＫＬ１言語

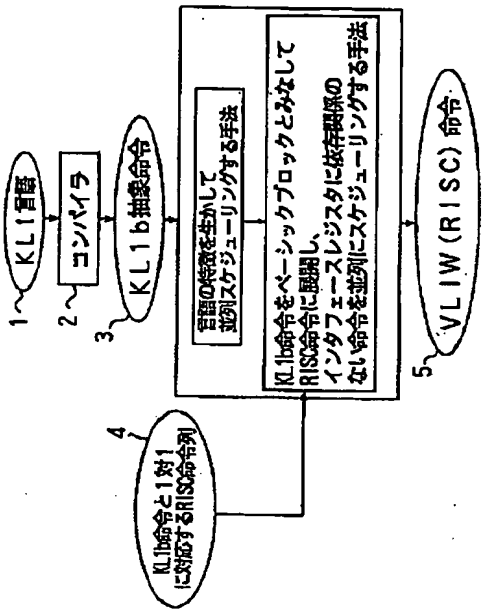
9

2　コンパイラ
3　ＫＬ１ｂ命令

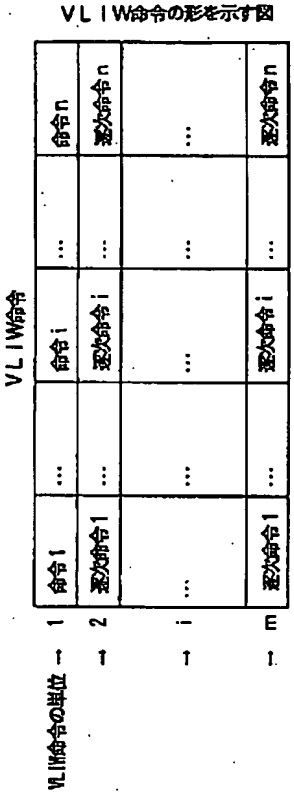
10

4　ＫＬ１ｂ命令と１対１に対応するＲＩＳＣ命令列
5　ＲＩＳＣ命令

【図１】

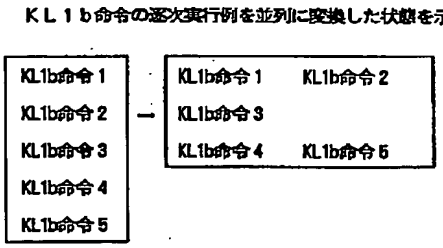


【図２】

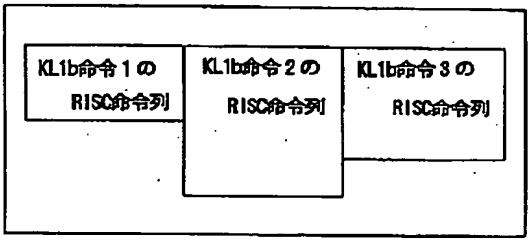


【図３】

【図４】

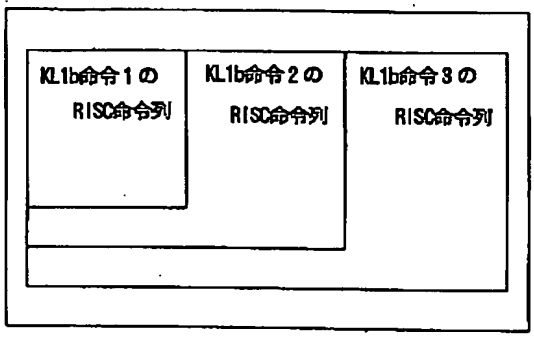


【図５】



【図６】

広げられたベーシックブロックの中のＲＩＳＣ命令を示す図



【図７】

実施例のコンパイル結果を示す図

```
label   append/3.
        try_me_else append/3/1.
        wait_list 1,append/3/1.
        read_car 1,4.
        read_cdr 1,1.
        put_list 5.
        write_car_value 5,4.
        write_cdr_variable 5,4.
        get_list_value 5,3.
        put_value 4,3.
        execute append/3.
label   append/3/1.
        try_me_else append/3/2.
        wait_constant [],1,append/3/2.
        get_value 2,3.
        proceed.
label   append/3/2.
        suspend append/3.
```

【図８】

実施例のＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図

KL1b 命令	RISC 命令列	
try_me_else →	exeg acp,(append/3/1,3),IM#1	parbox5cm
wait_list →	exti_u8 1,(Ai,label),IM#1	
	ref_reg 2,1	
	slct_m 3,2	
	extt_u8 4,3,IM#0	
	ceq_u32 5,4,IM#REF	ifundf 3
	br 5,T,PD(3,Ai)	cne_u32 6,4,IM#LIST
	br 6,T,SIU(Ai)	ind_mv 1,3

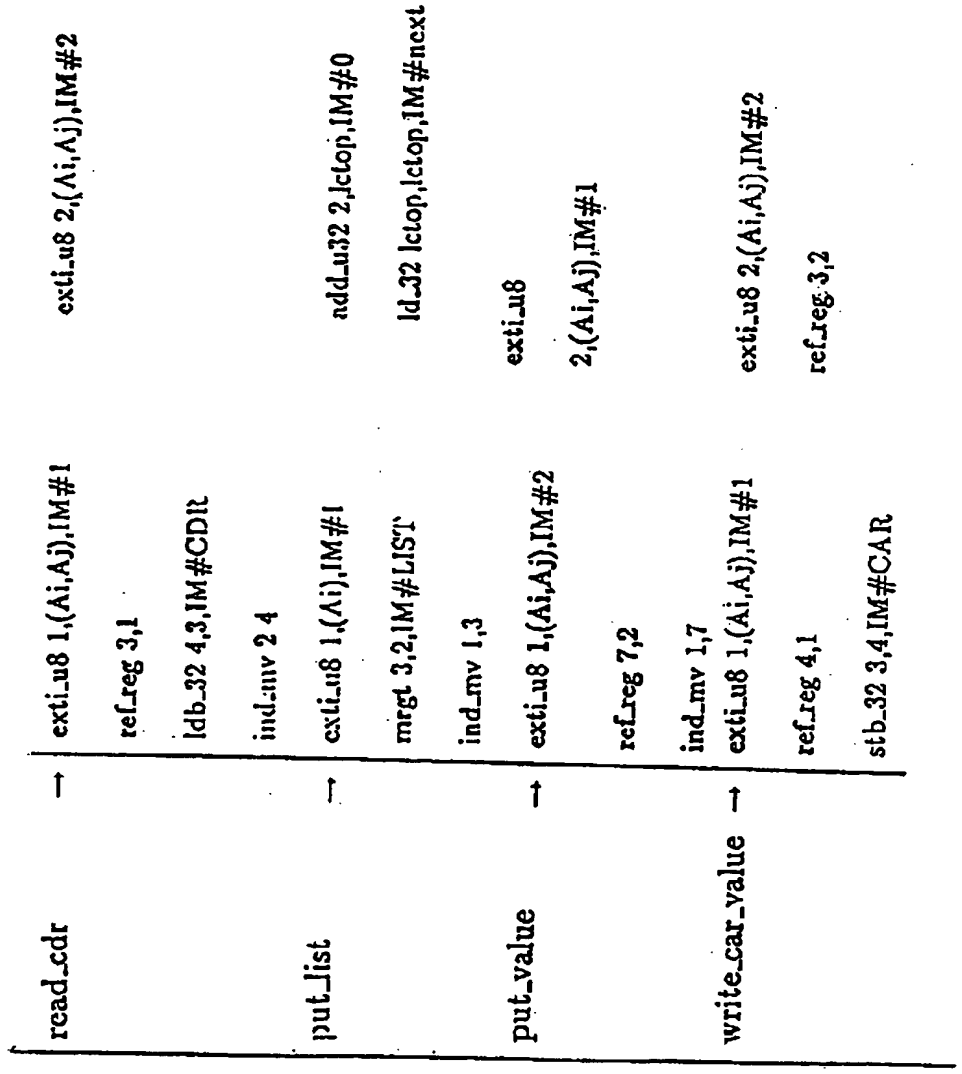
【図９】

実施例のＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図（続き）

wait_constant →	exti_u8 1,(C,Ai,label),IM#2	extia_u16 atom,(C,Ai),IM#1
	ref_reg 2,1	mrgt atom,atom IM#ATOM
	slct_m 5,2	
	extt_u8 3,5,IM#0	
	ceq_u32 4,3,IM#REF	ifundf 5
	br 4,T,PD(5,Ai)	ind_mv 1,5
	br 6,T,SIU(Ai)	cne_u32 6,5,atom
read_car ↑	exti_u8 1,(Ai,Aj),IM#1	exti_u8 2,(Ai,Aj),IM#2
	ref_reg 3,1	
	ldb_32 4,3,IM#CAR	
	ind_mv 2 4	

【図１０】

実施例のＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図（続き）



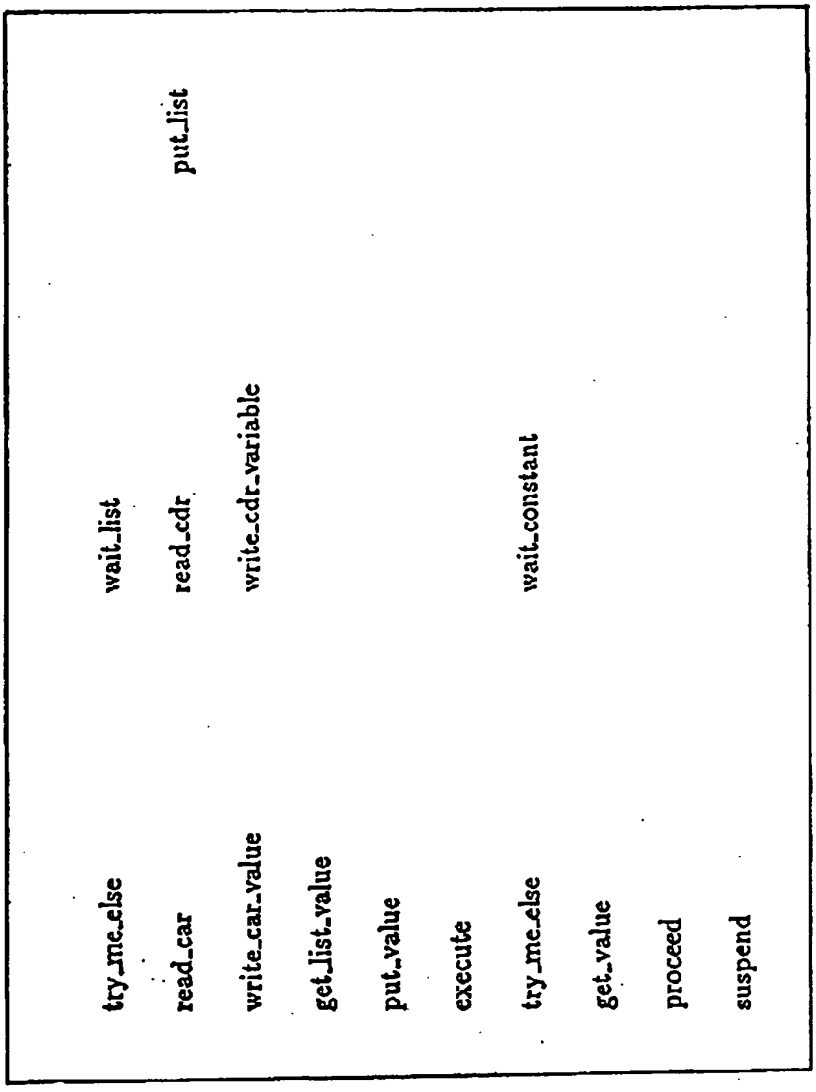
【図１１】

実施例のＫＬ１ｂ命令とＲＩＳＣ命令列の対を示す図（続き）

write_cdr_variable →	exti_u8 1,(Ai,Aj),IM#1	exti_u8 2,(Ai,Aj),IM#2	add_u32 3,grtop,IM#0
	ref_reg 4,1	mrgt 5,3,IM#REF	ld_32 grtop,grtop,IM#next
	std_32 3,5,IM#0		
	stb_32 5,4,IM#CDR	ind_mv 2,5	
get_list_value →	exti_u8 1,(Ai,Aj),IM#1	exti_u8 2,(Ai,Aj),IM#2	
	ref_reg 3,1	ref_reg 4,2	
	slct_m 5,4		
	extt_u8 6,5,IM#0		
	cne_u32 7,6,IM#UNDF		
	br 7,T,Getlv(Ai,5,Aj,PCT)		
	mrgt 8,5,IM#REF		
	stb_32 3,8,IM#0		
	ind_mv 2,8		
get_value →	br 0,T,Getv(Ai,Aj)		

【図１２】

実施例のＫＬ１ｂ命令を並列に記述した状態を示す図



【図１３】

実施例をＲＩＳＣ命令列に置き換え整理した状態を示す図

extg acp,(append/3/1,3),IM#1	exti_u8 1,(1,append/3/1),IM#1
ref_reg 2,1	
slct_rn 3,2	
extt_u8 4,3,IM#0	
ceq_u32 5,4,IM#REF	ifundf 3
br 5,T,PD(3,1)	cne_u32 6,4,IM#LIST
br 6,T,STU(1)	
exti_u8 1,(1,4),IM#1	exti_u8 2,(1,4),IM#2
ref_reg 3,1	exti_u8 6,(1,1),IM#2
ldb_32 4,3,IM#CAR	ref_reg 7,5
ind_mv 2,4	mrgt 11,10,IM#LIST
ldb_32 8,7,IM#CDR	ind_mv 9,11
ind_mv 6,8	
exti_u8 1,(5,4),IM#1	exti_u8 2,(5,4),IM#2
	exti_u8 5,(5,4),IM#1
	exti_u8 5,(1,1),IM#1
	exti_u8 9,(5),IM#1
	add_u32 10,lctop,IM#0
	ld_32 lctop,lctop,IM#next
	ind_mv 1,3

【図１４】

実施例をＲＩＳＣ命令列に置き換え整理した状態を示す図（続き）

ref_reg 4,1	ref_reg 3,2	exti_u8 6,(5,4),IM#2
stb_32 3,4,IM#CAR	add_u32 7,grtop,IM#0	
ref_reg 8,5	mrgt 9,7,IM#REF	ld_32 grtop,grtop,IM#next
std_32 7,9,IM#0		
stb_32 9,8,IM#CDR	ind_mv 6,9	
exti_u8 1,(5,3),IM#1	exti_u8 2,(5,3),IM#2	
ref_reg 3,1	ref_reg 4,2	
slct_m 5,4		
extt_u8 6,5,IM#0		
cne_u32 7,6,IM#UNDF		
br 7,T,Gelv(5,5,3,PCT)		
mrgt 8,5,IM#REF		
stb_32 3,8,IM#0		
ind_mv 2,8		

【図１５】

実施例をＲＩＳＣ命令列に置き換え整理した状態を示す図（続き）

```
exti_u8 1,(4,3),IM#2        exti_u8 2,(4,3),IM#1

ref_reg 7,2

ind_mv 1,7

exti_u8 argument_num,(append/3,3),IM#2        extg pct,(append/3,3),IM#1

label append/3/1

extg acp,(append/3/2,3),IM#1    exti_u8 1,([],1,append/3/2),IM#2    extia_u16 atom,([],1),IM#1

ref_reg 2,1                 mrgt atom,atom,IM#ATOM

slct_m 5,2

extt_u8 3,5,IM#0

ceq_u32 4,3,IM#REF          ifundf 5
```

【図１６】

実施例をＲＩＳＣ命令列に置き換え整理した状態を示す図（続き）